In [1]:
```
import xarray as xr
import numpy as np
import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

import torch
from torch import nn
```

In [2]:
```
class TemperatureLandcover1D(Dataset):
    def __init__(self, path, subset="train", seed=87, normalize=False, reduc
        self.path = path
        self.subset = subset
        self.seed = seed

        # Load the data
        ds = xr.open_dataset(self.path)
        t2m = ds.air_temperature_at_2_metres.values  # (143, 489, 529)
        t30m = ds.air_temperature_at_30_metres.values  # (143, 489, 529)
        landcover = ds.landcover.values  # (member, x, y) int8

        # To include all categories, get the landcover categories
        landcover_classes = np.unique(landcover)

        # Create a dictionary to map landcover categories to integers
        landcover_to_ix = {category: i for i, category in enumerate(landcove



        # Flatten the data
        t2m = t2m.flatten()
        t30m = t30m.flatten()

        # Replace the landcover variable categories to an array of integers
        landcover = np.array([landcover_to_ix[category] for category in lanc


        # Normalize the temperature data
        if normalize:
            t2m = (t2m - t2m.mean()) / t2m.std()
            t30m = (t30m - t30m.mean()) / t30m.std()

        # Split the data into training, testing, and validation subsets
        np.random.seed(self.seed)
        if reduce_to is None:
            reduce_to = t2m.size

        train_idxs = np.random.randint(0, t2m.size, reduce_to // 2)  # 50% i
        test_idxs = np.random.randint(0, t2m.size, reduce_to // 4)  # 25% in
        val_idxs = np.random.randint(0, t2m.size, reduce_to // 4)  # 25% in









        if self.subset == "train":
            idxs = train_idxs
        elif self.subset == "test":
            idxs = test_idxs
        elif self.subset == "val":
```

```
            idxs = val_idxs
        else:
            raise ValueError("Unknown subset")


        # Convert the selected indices to tensors
        # ensure idxs variable is within the bounds of the landcover array b
        # modulus operator (%) to ensure that the index is within the range
        self.t2m = torch.tensor(t2m[idxs % t2m.size])
        self.t30m = torch.tensor(t30m[idxs % t30m.size])
        self.landcover = torch.tensor(landcover[idxs % landcover.size])

    def __len__(self):
        return len(self.t2m)

    def __getitem__(self, idx):
        """Return an item of the dataset.

        Parameters
        ----------
        idx: int
            Index of the item to return

        Returns
        -------
        x: float tensor of shape (1,)
            Temperature at 2m
        c: int64
            Landcover category
        y: float tensor of shape (1,)
            Temperature at 30m
        """
        x = self.t30m[idx].unsqueeze(0)
        c = self.landcover[idx].long()
        y = self.t2m[idx].unsqueeze(0)
        return x, c, y
```

In [ ]:

In [3]:
```
ds = TemperatureLandcover1D("/Users/marwa/Desktop/MyWorkingdirectory/interns

x, c, y = ds[0]
x
```

Out[3]:
```
tensor([280.3035])
```

In [4]:
```
class LinearRegressionWithEmbedding(nn.Module):
    def __init__(self, n_landcovers=33, embedding_size=3, output_size=1):
        super().__init__()
        self.landcover_embedding = nn.Embedding(n_landcovers, embedding_size
        self.linear = nn.Linear(embedding_size + 1, output_size)

    def forward(self, x, c):
        emb = self.landcover_embedding(c)
        x = torch.cat([x, emb], dim=1)
        return self.linear(x)
```

In [ ]:

In [5]:
```
# Load dataset
# ------------
#specify batch size
```

```
batch_size = 4096

#specify file path
ncfile = "/Users/marwa/Desktop/MyWorkingdirectory/internship-marwa/data/mera

#The TemperatureLandcover1D class is used to create instances of the dataset
#The subset parameter is set to "train" for the training data and "test" for
#The normalize parameter is set to True for both datasets.
#The reduce_to parameter is set to 3,000,000 for both datasets.

training_data = TemperatureLandcover1D(
    ncfile, subset="train", normalize=True, reduce_to=3_000_000
)
test_data = TemperatureLandcover1D(
    ncfile, subset="test", normalize=True, reduce_to=3_000_000
)

x0, c0, y0 = training_data[0]
print(f"The dataset has {len(training_data)} items. Each item looks like {x0


train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

x, c, y = next(iter(train_dataloader))
print(f"The dataloader create batches of items of shape {x.shape, c.shape, y


# Network
# -------
net = LinearRegressionWithEmbedding()
y_pred = net(x, c)


# Loss and optimizer
# ------------------
loss_fn = torch.nn.MSELoss()
print(f"Loss value: {loss_fn(y_pred, y)}")

optimizer = torch.optim.SGD(net.parameters(), lr=1e-3)


# Training
# --------
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")


def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader)
    model.train()
    for batch, (x, c, y) in enumerate(dataloader):
        x, c, y = x.to(device), c.to(device), y.to(device)

        # Compute prediction error
        pred = model(x, c)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
```

```
            loss, current = loss.item(), batch
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test(dataloader, model, loss_fn):
    num_batches = len(dataloader)
    model.eval()
    test_loss = 0
    baseline = 0
    with torch.no_grad():
        for x, c, y in dataloader:
            x, c, y = x.to(device), c.to(device), y.to(device)
            pred = model(x, c)
            test_loss += loss_fn(pred, y).item()
            baseline += loss_fn(x, y).item()

    test_loss /= num_batches
    print(
        f"Test Error: \n Mean squared error: {test_loss:>8f} (baseline: {bas
    )


epochs = 2
for t in range(epochs):
    print(f"\nEpoch {t+1}\n-------------------------------")
    train(train_dataloader, net, loss_fn, optimizer)
    test(test_dataloader, net, loss_fn)
print("Done!")
```

```
The dataset has 1500000 items. Each item looks like (tensor([-0.6996]), ten
sor(0), tensor([-0.5873]))
The dataloader create batches of items of shape (torch.Size([4096, 1]), tor
ch.Size([4096]), torch.Size([4096, 1]))
Loss value: 0.7830725908279419
Using cpu device

Epoch 1
-------------------------------
loss: 0.783073  [    0/  367]
loss: 0.471612  [  100/  367]
loss: 0.284469  [  200/  367]
loss: 0.185182  [  300/  367]
Test Error:
 Mean squared error: 0.141009 (baseline: 2.821028)


Epoch 2
-------------------------------
loss: 0.139461  [    0/  367]
loss: 0.103497  [  100/  367]
loss: 0.070247  [  200/  367]
loss: 0.053444  [  300/  367]
Test Error:
 Mean squared error: 0.045604 (baseline: 2.821028)

Done!
```

In [ ]:

In [6]:
```
class LinearRegressionWithEmbedding(nn.Module):
    def __init__(self, n_landcovers=33, embedding_size=3, output_size=1):
        super().__init__()
        self.landcover_embedding = nn.Embedding(n_landcovers, embedding_size
        self.fc1 = nn.Linear(embedding_size+1, 128)
```

```
        self.fc2 = nn.Linear(128, output_size)

    def forward(self, x, c):
        emb = self.landcover_embedding(c)
        x = torch.cat([x, emb], dim=1)
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

In [7]:

```
# Load dataset
# ------------
#specify batch size
batch_size = 4096

#specify file path
ncfile = "/Users/marwa/Desktop/MyWorkingdirectory/internship-marwa/data/mera

#The TemperatureLandcover1D class is used to create instances of the dataset
#The subset parameter is set to "train" for the training data and "test" for
#The normalize parameter is set to True for both datasets.
#The reduce_to parameter is set to 3,000,000 for both datasets.

training_data = TemperatureLandcover1D(
    ncfile, subset="train", normalize=True, reduce_to=3_000_000
)
test_data = TemperatureLandcover1D(
    ncfile, subset="test", normalize=True, reduce_to=3_000_000
)

x0, c0, y0 = training_data[0]
print(f"The dataset has {len(training_data)} items. Each item looks like {x0


train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

x, c, y = next(iter(train_dataloader))
print(f"The dataloader create batches of items of shape {x.shape, c.shape, y


# Network
# -------
net = LinearRegressionWithEmbedding()
y_pred = net(x, c)


# Loss and optimizer
# -------------------
loss_fn = torch.nn.MSELoss()
print(f"Loss value: {loss_fn(y_pred, y)}")

optimizer = torch.optim.SGD(net.parameters(), lr=1e-3)


# Training
# --------
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")


def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader)
    model.train()
    for batch, (x, c, y) in enumerate(dataloader):
        x, c, y = x.to(device), c.to(device), y.to(device)
```

```
        # Compute prediction error
        pred = model(x, c)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), batch
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test(dataloader, model, loss_fn):
    num_batches = len(dataloader)
    model.eval()
    test_loss = 0
    baseline = 0
    with torch.no_grad():
        for x, c, y in dataloader:
            x, c, y = x.to(device), c.to(device), y.to(device)
            pred = model(x, c)
            test_loss += loss_fn(pred, y).item()
            baseline += loss_fn(x, y).item()

    test_loss /= num_batches
    print(
        f"Test Error: \n Mean squared error: {test_loss:>8f} (baseline: {bas
    )


epochs = 2
for t in range(epochs):
    print(f"\nEpoch {t+1}\n-------------------------------")
    train(train_dataloader, net, loss_fn, optimizer)
    test(test_dataloader, net, loss_fn)
print("Done!")
```

```
The dataset has 1500000 items. Each item looks like (tensor([-0.6996]), ten
sor(0), tensor([-0.5873]))
The dataloader create batches of items of shape (torch.Size([4096, 1]), tor
ch.Size([4096]), torch.Size([4096, 1]))
Loss value: 1.0053765773773193
Using cpu device

Epoch 1
-------------------------------
loss: 1.005377  [    0/  367]
loss: 0.277922  [  100/  367]
loss: 0.086489  [  200/  367]
loss: 0.046113  [  300/  367]
Test Error:
 Mean squared error: 0.037214 (baseline: 2.821028)


Epoch 2
-------------------------------
loss: 0.033996  [    0/  367]
loss: 0.034278  [  100/  367]
loss: 0.029303  [  200/  367]
loss: 0.027222  [  300/  367]
Test Error:
 Mean squared error: 0.027144 (baseline: 2.821028)

Done!
```

In [ ]:

In [ ]:

In [ ]: